Министерство образования Калининградской области

государственное бюджетное учреждение Калининградской области

профессиональная образовательная организация

«Колледж информационных технологий и строительства»

(ГБУ КО ПОО «КИТиС»)

**Отчет по учебной практике**

УП.01 Разработка модулей программного обеспечения для компьютерных систем

по ПМ.01 Разработка модулей программного обеспечения для компьютерных систем

Специальность 09.02.07 «Информационные системы и программирование»

Сроки прохождения практики:

с «09» сентября 2024 г. по «09» ноября 2024 г.

Место практики ГБУ КО ПОО «КИТиС»

|  |  |
| --- | --- |
| Выполнил: | студент 4 курса,  группы ИСп 21-2к  Пухно Дмитрий Алексеевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Проверила: | Большакова-Стрекалова Анна Викторовна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Калининград, 2024

Содержание

[Введение 3](#_Toc182782182)

[1.Разработка программных модулей 4](#_Toc182782183)

[1.1 Задание 1 4](#_Toc182782184)

[1.2 Задание 2 6](#_Toc182782185)

[1.3 Задание 3 11](#_Toc182782186)

[1.4 Задание 4 17](#_Toc182782187)

[1.5 Задание 5 19](#_Toc182782188)

[1.7 Задание 7 22](#_Toc182782189)

[1.8 Задание 8 25](#_Toc182782190)

[1.9 Задание 9 32](#_Toc182782191)

[2. Разработка АИС 42](#_Toc182782192)

[3.1 Описание и цели системы 42](#_Toc182782193)

[3.2 Функциональные требования 42](#_Toc182782194)

[3.3 Нефункциональные требования 43](#_Toc182782195)

[3.4 Структура системы и описание классов 43](#_Toc182782196)

[3.5 Алгоритмы и описание действий пользователя 44](#_Toc182782197)

[3.6 Тестирование системы 44](#_Toc182782198)

[3.7 Техническое задание (ТЗ) 45](#_Toc182782199)

[3.8 Руководство пользователя 46](#_Toc182782200)

[3.9 Протокол тестирования 46](#_Toc182782201)

[4. Разработка Базы Данных 47](#_Toc182782202)

[5. Разработка 1С «Издательство» 57](#_Toc182782203)

[6. Разработка мобильного приложения 59](#_Toc182782204)

[Заключение 61](#_Toc182782205)

[Список используемой литературы 62](#_Toc182782206)

# Введение

Данный отчёт служит для представления результатов прохождения учебной практики по программированию. Она состоит из пяти дисциплин разработка программных модулей, разработка мобильных приложений, поддержка и тестирование программных модулей, программирование 1С, системное программирование. Так же практика включает создание собственного WEB-сайта.

Цели:

1. Расширить свои познания и улучшить практические навыки по дисциплинам;

2. Создать собственный сайт для демонстрации выполненной работы;

3. Решить поставленные задачи для проверки усвоения материала дисциплин;

# 1.Разработка программных модулей

## 1.1 Задание 1

14. Округлить все элементы массива до целого числа

Таблица 1 – Задание 1

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| matrix | double[][] | Исходная матрица с элементами типа double, которая будет округляться. |
| roundedMatrix | int[][] | Новая матрица с элементами типа int, полученная после округления значений из matrix. |
| rows | int | Количество строк в матрице matrix. |
| cols | int | Количество столбцов в матрице matrix. |
| i | int | |  | | --- | |  |  |  | | --- | | Индекс строки, используемый в циклах для обхода матрицы. | |
| j | int | Индекс столбца, используемый в циклах для обхода матрицы. |
| elem | int | Элемент матрицы, используемый в методе printMatrix для вывода значений. |

public class MatrixRounding {  
 public static void main(String[] args) {  
 // Пример матрицы с элементами типа double  
 double[][] matrix = {  
 {1.2, 3.7, 4.5},  
 {5.6, 7.1, 8.9},  
 {9.0, 0.4, 2.3}  
 };  
  
 // Округляем элементы матрицы  
 int[][] roundedMatrix = *roundMatrix*(matrix);  
  
 // Выводим округленную матрицу  
 *printMatrix*(roundedMatrix);  
 }  
  
 // Метод для округления элементов матрицы  
 public static int[][] roundMatrix(double[][] matrix) {  
 int rows = matrix.length;  
 int cols = matrix[0].length;  
 int[][] roundedMatrix = new int[rows][cols];  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 roundedMatrix[i][j] = (int) Math.*round*(matrix[i][j]);  
 }  
 }  
  
 return roundedMatrix;  
 }  
  
 // Метод для вывода матрицы  
 public static void printMatrix(int[][] matrix) {  
 for (int[] row : matrix) {  
 for (int elem : row) {  
 System.*out*.print(elem + " ");  
 }  
 System.*out*.println();  
 }  
 }  
}

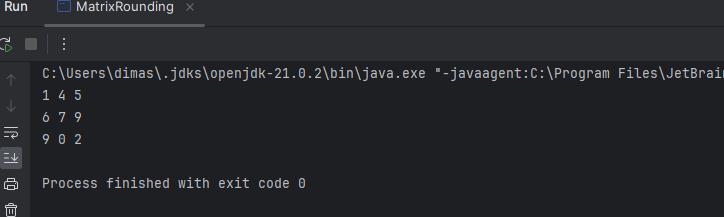


Рисунок 1 – Результат работы задания 1

## 1.2 Задание 2

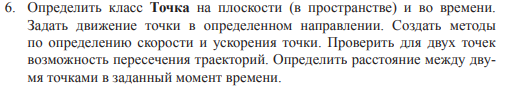


Таблица 2 – Задание 2.

|  |  |  |
| --- | --- | --- |
| **Переменная** | Тип данных | Описание |
| x | Double | Координата x точки |
| y | Double | Координата y точки |
| z | Double | Координата z точки |
| t | Double | |  | | --- | |  |  |  | | --- | | Время точки. | |
| vx | Double | Скорость точки по координате x |
| vy | Double | Скорость точки по координате y |
| vz | Double | Скорость точки по координате z |
| ax | Double | Ускорение точки по координате x |
| ay | Double | Ускорение точки по координате y |
| az | Double | Ускорение точки по координате z |
| dt | double | Изменение времени для обновления координат и проверки пересечения траекторий. |

public class Point {

private double x; // Координата x точки  
 private double y; // Координата y точки  
 private double z; // Координата z точки  
 private double t; // Время точки  
  
 private double vx; // Скорость по x  
 private double vy; // Скорость по y  
 private double vz; // Скорость по z  
  
 private double ax; // Ускорение по x  
 private double ay; // Ускорение по y  
 private double az; // Ускорение по z  
  
 // Конструктор для создания точки с начальным положением и скоростью  
 public Point(double x, double y, double z, double t, double vx, double vy, double vz) {  
 this.x = x;  
 this.y = y;  
 this.z = z;  
 this.t = t;  
 this.vx = vx;  
 this.vy = vy;  
 this.vz = vz;  
 }  
  
 // Метод для задания ускорения  
 public void setAcceleration(double ax, double ay, double az) {  
 this.ax = ax;  
 this.ay = ay;  
 this.az = az;  
 }  
  
 // Метод для обновления координат точки на основе времени  
 public void updatePosition(double dt) {  
 // Обновляем скорость  
 vx += ax \* dt;  
 vy += ay \* dt;  
 vz += az \* dt;  
  
 // Обновляем координаты  
 x += vx \* dt + 0.5 \* ax \* dt \* dt;  
 y += vy \* dt + 0.5 \* ay \* dt \* dt;  
 z += vz \* dt + 0.5 \* az \* dt \* dt;  
  
 // Обновляем время  
 t += dt;  
 }  
  
 // Метод для вычисления скорости точки  
 public double[] getVelocity() {  
 return new double[]{vx, vy, vz};  
 }  
  
 // Метод для вычисления ускорения точки  
 public double[] getAcceleration() {  
 return new double[]{ax, ay, az};  
 }  
  
 // Метод для определения расстояния между двумя точками в заданный момент времени  
 public static double distance(Point p1, Point p2) {  
 double dx = p1.x - p2.x;  
 double dy = p1.y - p2.y;  
 double dz = p1.z - p2.z;  
 return Math.*sqrt*(dx \* dx + dy \* dy + dz \* dz);  
 }  
  
 // Метод для проверки возможности пересечения траекторий двух точек  
 public static boolean canIntersect(Point p1, Point p2, double dt) {  
 double[] v1 = p1.getVelocity();  
 double[] v2 = p2.getVelocity();  
  
 double x1 = p1.x + v1[0] \* dt;  
 double y1 = p1.y + v1[1] \* dt;  
 double z1 = p1.z + v1[2] \* dt;  
  
 double x2 = p2.x + v2[0] \* dt;  
 double y2 = p2.y + v2[1] \* dt;  
 double z2 = p2.z + v2[2] \* dt;  
  
 return (x1 == x2 && y1 == y2 && z1 == z2);  
 }  
  
 public static void main(String[] args) {  
 Point p1 = new Point(0, 0, 0, 0, 1, 1, 1);  
 Point p2 = new Point(1, 1, 1, 0, -1, -1, -1);  
  
 p1.setAcceleration(0, 0, 0);  
 p2.setAcceleration(0, 0, 0);  
  
 p1.updatePosition(1);  
 p2.updatePosition(1);  
  
 System.*out*.println("Distance between p1 and p2: " + Point.*distance*(p1, p2));  
  
 boolean intersect = Point.*canIntersect*(p1, p2, 1);  
 System.*out*.println("Do the trajectories intersect? " + intersect);  
 }  
}



Рисунок 2 – Результат работы задания 2.

## 1.3 Задание 3

8. Создать обьект класса Пианино, используя классы Клавиша, Педаль. Методы: настроить, играть на пианино, нажимать клавишу.

Таблица 3 – Задание 3.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| note | String | Нота, соответствующая клавише пианино. |
| pressed | boolean | Флаг, указывающий, нажата ли клавиша или педаль (true или false). |
| name | String | Название педали (например, "правая педаль", "левая педаль"). |
| keys | Key[] | Массив из 88 объектов класса Key, представляющий клавиши пианино. |
| rightPedal | Pedal | Объект класса Pedal, представляющий правую педаль. |
| leftPedal | Pedal | Объект класса Pedal, представляющий левую педаль. |
| tuned | boolean | Флаг, указывающий, настроено ли пианино (true или false). |
| keyNumber | int | Номер клавиши, на которую нажимают (в диапазоне от 1 до 88). |
| pedalName | String | Название педали, на которую нажимают ("right" или "left"). |

class Key {  
 private String note;  
 private boolean pressed;  
  
 public Key(String note) {  
 this.note = note;  
 this.pressed = false;  
 }  
  
 public void press() {  
 this.pressed = true;  
 System.*out*.println("Клавиша " + this.note + " нажата.");  
 }  
  
 public void release() {  
 this.pressed = false;  
 System.*out*.println("Клавиша " + this.note + " отпущена.");  
 }  
}  
class Pedal {  
 private String name;  
 private boolean pressed;  
  
 public Pedal(String name) {  
 this.name = name;  
 this.pressed = false;  
 }  
  
 public void press() {  
 this.pressed = true;  
 System.*out*.println("Педаль '" + this.name + "' нажата.");  
 }  
  
 public void release() {  
 this.pressed = false;  
 System.*out*.println("Педаль '" + this.name + "' отпущена.");  
 }  
}  
class Piano {  
 private Key[] keys;  
 private Pedal rightPedal;  
 private Pedal leftPedal;  
 private boolean tuned;  
  
 public Piano() {  
 this.keys = new Key[88];  
 for (int i = 0; i < 88; i++) {  
 this.keys[i] = new Key("Нота " + (i + 1));  
 }  
 this.rightPedal = new Pedal("правая педаль");  
 this.leftPedal = new Pedal("левая педаль");  
 this.tuned = false;  
 }  
  
 public void tune() {  
 this.tuned = true;  
 System.*out*.println("Пианино настроено.");  
 }  
  
 public void pressKey(int keyNumber) {  
 if (!this.tuned) {  
 System.*out*.println("Пианино не настроено!");  
 return;  
 }  
  
 if (keyNumber >= 1 && keyNumber <= 88) {  
 this.keys[keyNumber - 1].press();  
 } else {  
 System.*out*.println("Неверный номер клавиши!");  
 }  
 }  
  
 public void releaseKey(int keyNumber) {  
 if (keyNumber >= 1 && keyNumber <= 88) {  
 this.keys[keyNumber - 1].release();  
 } else {  
 System.*out*.println("Неверный номер клавиши!");  
 }  
 }  
  
 public void pressPedal(String pedalName) {  
 if ("right".equals(pedalName)) {  
 this.rightPedal.press();  
 } else if ("left".equals(pedalName)) {  
 this.leftPedal.press();  
 } else {  
 System.*out*.println("Неверная педаль!");  
 }  
 }  
 public void releasePedal(String pedalName) {  
 if ("right".equals(pedalName)) {  
 this.rightPedal.release();  
 } else if ("left".equals(pedalName)) {  
 this.leftPedal.release();  
 } else {  
 System.*out*.println("Неверная педаль!");  
 }  
 }  
 public void play() {  
 if (!this.tuned) {  
 System.*out*.println("Пианино не настроено! Невозможно играть.");  
 return;  
 }  
 System.*out*.println("Вы играете на пианино.");  
 }  
}  
public class Main {  
 public static void main(String[] args) {  
 Piano piano = new Piano();  
  
 piano.tune(); // Настроить пианино  
  
 piano.pressKey(40); // Нажать клавишу  
 piano.releaseKey(40); // Отпустить клавишу  
  
 piano.pressPedal("right"); // Нажать правую педаль  
 piano.releasePedal("right"); // Отпустить правую педаль  
  
 piano.play(); // Играть на пианино  
 }  
}

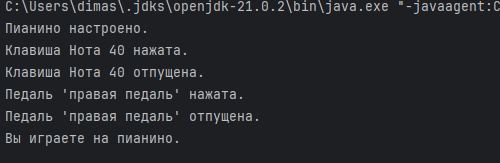


Рисунок 3 – Результат работы задания 3.

## 1.4 Задание 4

18. Создать класс Программа Передач с внутренним классом, с помощью обьектов которого можно хранить информацию о названии телеканалов и программ.

Таблица 4 – задание 4

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| channelName | String | Название телеканала (например, "Канал 1"). |
| programName | String | Название программы на данном канале (например, "Новости"). |
| schedule | TVSchedule | Объект класса TVSchedule, представляющий программу передач. |
| program1 | ChannelProgram | Объект внутреннего класса ChannelProgram, представляющий программу "Новости" на "Канале 1". |
| program2 | ChannelProgram | Объект внутреннего класса ChannelProgram, представляющий программу "Фильм" на "Канале 2". |

public class TVSchedule {

public class ChannelProgram {

private String channelName; // Название телеканала

private String programName; // Название программы

public ChannelProgram(String channelName, String programName) {

this.channelName = channelName;

this.programName = programName;

}

public String getChannelName() {

return channelName;

}

public String getProgramName() {

return programName;

}

public void showInfo() {

System.out.println("Канал: " + channelName + ", Программа: " + programName);

}

}

public ChannelProgram addProgram(String channelName, String programName) {

return new ChannelProgram(channelName, programName);

}

public static void main(String[] args) {

TVSchedule schedule = new TVSchedule();

ChannelProgram program1 = schedule.addProgram("Канал 1", "Новости");

ChannelProgram program2 = schedule.addProgram("Канал 2", "Фильм");

program1.showInfo();

program2.showInfo();

}

}

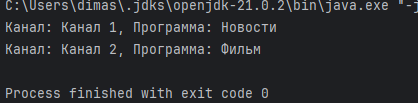


Рисунок 4 – результат работы задания 4

## 1.5 Задание 5

18. В тексте определить все согласные буквы, встречающиеся не более чем в двух словах.

Таблица 5 – Задание 5.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| text | String | Исходный текст для анализа. |
| consonants | Set<Character> | Множество согласных букв, найденных в слове. |
| consonantFrequency | Map<Character, Integer> | Хранит частоту появления каждой согласной в разных словах. |
| uniqueWords | Set<String> | Множество уникальных слов текста. |
| result | Set<Character> | Множество согласных, встречающихся не более чем в двух словах текста. |
| rareConsonants | Set<Character> | Результат программы — согласные, встречающиеся не более чем в двух словах. |

import java.util.\*;  
  
public class ConsonantsInText {  
 private static Set<Character> getConsonants(String word) {  
 Set<Character> consonants = new HashSet<>();  
 String lowerWord = word.toLowerCase();  
 for (char c : lowerWord.toCharArray()) {  
 if (*isConsonant*(c)) {  
 consonants.add(c);  
 }  
 }  
 return consonants;  
 }  
 private static boolean isConsonant(char c) {  
 return "бвгджзйклмнпрстфхцчшщ".indexOf(c) != -1;  
 }  
 public static Set<Character> findRareConsonants(String text) {  
 Map<Character, Integer> consonantFrequency = new HashMap<>();  
 Set<String> uniqueWords = new HashSet<>(Arrays.*asList*(text.toLowerCase().split("\\s+")));  
 for (String word : uniqueWords) {  
 Set<Character> consonantsInWord = *getConsonants*(word);  
 for (char consonant : consonantsInWord) {  
 consonantFrequency.put(consonant, consonantFrequency.getOrDefault(consonant, 0) + 1);  
 }  
 }  
 Set<Character> result = new HashSet<>();  
 for (Map.Entry<Character, Integer> entry : consonantFrequency.entrySet()) {  
 if (entry.getValue() <= 2) {  
 result.add(entry.getKey());  
 }  
 }  
  
 return result;  
 }  
 public static void main(String[] args) {  
 String text = "Антифриз — общее название для жидкостей, не замерзающих при низких температурах. Применяются в установках, работающих при низких температурах, для охлаждения двигателей внутреннего сгорания, в качестве авиационных противообледенительных жидкостей и в качестве средства для очистки стекол.";  
  
 // Находим согласные, встречающиеся не более чем в двух словах  
 Set<Character> rareConsonants = *findRareConsonants*(text);  
  
 // Вывод результата  
 System.*out*.println("Согласные, встречающиеся не более чем в двух словах: " + rareConsonants);  
 }  
}

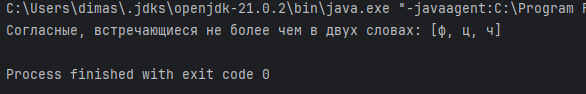


Рисунок 5 – Результат работы задания 5.

## 1.7 Задание 7

18. В правильной шестиугольной призме провести сечение, проходящее через одну из сторон нижнего основание и противоположенную сторону верхнего основания.

public class HexagonalPrism {  
 // Длина стороны правильного шестиугольника  
 private double sideLength;  
 // Высота призмы  
 private double height;  
  
 public HexagonalPrism(double sideLength, double height) {  
 this.sideLength = sideLength;  
 this.height = height;  
 }  
  
 // Метод для вычисления площади сечения  
 public double getSectionArea() {  
 // Сечение представляет собой прямоугольник со сторонами:  
 // длина стороны шестиугольника и высота призмы  
 double sectionArea = sideLength \* height;  
 return sectionArea;  
 }  
  
 public static void main(String[] args) {  
 // Пример использования  
 double sideLength = 5.0; // длина стороны правильного шестиугольника  
 double height = 10.0; // высота призмы  
  
 HexagonalPrism prism = new HexagonalPrism(sideLength, height);  
  
 // Вывод площади сечения  
 System.*out*.println("Площадь сечения: " + prism.getSectionArea());  
 }  
}

Таблица 6 – Задание 6.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| sideLength | double | Длина стороны правильного шестиугольника (одинакова для верхнего и нижнего основания призмы). |
| height | double | Высота призмы — расстояние между верхним и нижним основаниями. |
| sectionArea | double | Площадь сечения призмы, вычисляется как произведение длины стороны на высоту призмы. |
| prism | HexagonalPrism | Объект класса HexagonalPrism, представляющий правильную шестиугольную призму. |



Рисунок 7 – Результат работы задания 7.

## 1.8 Задание 8

18. Составить программу обработки двунаправленных связанных списков. (Связанный список данных состоит из указателей на

его начало и конец, а каждый элемент такого списка представляет

собой реализацию отдельного объекта.) При этом обеспечить возможность следующих операций:

* создание связанного списка (выделение для него памяти);
* уничтожение связанного списка (освобождение используемой
* памяти);
* инициализация связанного списка;
* деинициализация связанного списка;
* вставка элемента в середину связанного списка перед существующим элементом;
* присоединение элемента к концу связанного списка;
* Удаление элемента из связанного списка;
* возвращение первого элемента связанного списка;
* возвращение последнего элемента связанного списка.

// Класс для представления узла двусвязного списка  
class Node {  
 int data;  
 Node prev;  
 Node next;  
  
 public Node(int data) {  
 this.data = data;  
 this.prev = null;  
 this.next = null;  
 }  
}  
  
// Класс для работы с двусвязным списком  
class DoublyLinkedList {  
 Node head; // указатель на начало списка  
 Node tail; // указатель на конец списка  
  
 // Конструктор: создание пустого списка  
 public DoublyLinkedList() {  
 this.head = null;  
 this.tail = null;  
 }  
  
 // Операция 1: Инициализация (добавление элементов в начало списка)  
 public void initialize(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = tail = newNode;  
 } else {  
 newNode.next = head;  
 head.prev = newNode;  
 head = newNode;  
 }  
 }  
  
 // Операция 2: Деинициализация (очистка списка)  
 public void deinitialize() {  
 head = tail = null; // Очищаем список  
 }  
  
 // Операция 3: Вставка элемента в середину перед существующим элементом  
 public void insertBefore(int existingData, int newData) {  
 Node current = head;  
 while (current != null && current.data != existingData) {  
 current = current.next;  
 }  
 if (current != null) { // Нашли элемент  
 Node newNode = new Node(newData);  
 newNode.next = current;  
 newNode.prev = current.prev;  
  
 if (current.prev != null) {  
 current.prev.next = newNode;  
 } else {  
 head = newNode; // Если вставка в начало  
 }  
 current.prev = newNode;  
 }  
 }  
  
 // Операция 4: Присоединение элемента к концу списка  
 public void append(int data) {  
 Node newNode = new Node(data);  
 if (tail == null) {  
 head = tail = newNode;  
 } else {  
 tail.next = newNode;  
 newNode.prev = tail;  
 tail = newNode;  
 }  
 }  
  
 // Операция 5: Удаление элемента из списка  
 public void delete(int data) {  
 Node current = head;  
 while (current != null && current.data != data) {  
 current = current.next;  
 }  
 if (current != null) { // Нашли элемент  
 if (current.prev != null) {  
 current.prev.next = current.next;  
 } else {  
 head = current.next; // Удаление из начала списка  
 }  
 if (current.next != null) {  
 current.next.prev = current.prev;  
 } else {  
 tail = current.prev; // Удаление из конца списка  
 }  
 }  
 }  
  
 // Операция 6: Возвращение первого элемента списка  
 public int getFirst() {  
 if (head != null) {  
 return head.data;  
 } else {  
 throw new IllegalStateException("Список пуст");  
 }  
 }  
  
 // Операция 7: Возвращение последнего элемента списка  
 public int getLast() {  
 if (tail != null) {  
 return tail.data;  
 } else {  
 throw new IllegalStateException("Список пуст");  
 }  
 }  
  
 // Печать списка для проверки  
 public void printList() {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " ");  
 current = current.next;  
 }  
 System.*out*.println();  
 }  
}  
  
// Тестирование программы  
public class Main {  
 public static void main(String[] args) {  
 DoublyLinkedList list = new DoublyLinkedList();  
  
 // Инициализация списка  
 list.initialize(10);  
 list.initialize(20);  
 list.initialize(30);  
  
 System.*out*.println("Список после инициализации:");  
 list.printList();  
  
 // Присоединение элемента к концу списка  
 list.append(40);  
 System.*out*.println("Список после добавления элемента в конец:");  
 list.printList();  
  
 // Вставка элемента перед 20  
 list.insertBefore(20, 15);  
 System.*out*.println("Список после вставки 15 перед 20:");  
 list.printList();  
  
 // Удаление элемента 10  
 list.delete(10);  
 System.*out*.println("Список после удаления 10:");  
 list.printList();  
  
 // Возвращение первого и последнего элементов  
 System.*out*.println("Первый элемент списка: " + list.getFirst());  
 System.*out*.println("Последний элемент списка: " + list.getLast());  
  
 // Деинициализация списка  
 list.deinitialize();  
 System.*out*.println("Список после деинициализации:");  
 list.printList(); // ничего не выведет, так как список пуст  
 }  
}

Таблица 7 – Задание 7.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| Node(int data) | Node | Конструктор для создания узла с заданными данными. |
| initialize(int data) | void | Инициализация списка, добавление элемента в начало списка. |
| deinitialize() | void | Деинициализация списка, освобождение всех узлов списка (очистка списка). |
| insertBefore(int existingData, int newData) | void | Вставка нового узла перед существующим элементом в списке. |
| append(int data) | void | Добавление элемента в конец списка. |
| delete(int data) | void | Удаление узла из списка, если найден элемент с заданными данными. |
| getFirst() | int | Возвращает данные первого элемента списка. Если список пуст — бросает исключение. |
| getLast() | int | Возвращает данные последнего элемента списка. Если список пуст — бросает исключение. |
| printList() | void | Выводит на экран все элементы списка (для проверки текущего состояния списка). |

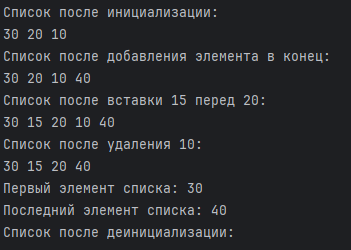


Рисунок 8 – Результат работы задания 8.

## 1.9 Задание 9

Итальянская игра «Математико».

Имеется квадратное поле из 25 клеток и набор из 52 карточек,

на которых записаны числа от 1 до 13, причем карточки с каждым

из этих чисел встречаются по четыре раза.

Разработать программу, которая позволит имитировать игру человека с компьютером: случайным образом извлекается какая-либо

из имеющихся карточек и выдается записанное на ней число. Каждый игрок заносит это число в одну из клеток квадрата, и так продолжается до тех пор, пока не будут заполнены все клетки квадрата.

import java.util.Arrays;  
import java.util.Random;  
import java.util.Scanner;  
  
public class MathGame {  
 static final int *SIZE* = 5; // Размер игрового поля (5x5)  
 static final int[] *NUMBERS* = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // Числа от 0 до 13  
 static int[][] *field* = new int[*SIZE*][*SIZE*]; // Поле игры  
 static boolean[][] *filled* = new boolean[*SIZE*][*SIZE*]; // Заполненные клетки  
  
 // Таблица 3: Количество чисел в каждой группе  
 static final int[] *counts* = {4, 4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 1};  
  
 static int[] *remainingCounts* = Arrays.*copyOf*(*counts*, *counts*.length); // Оставшиеся количества для чисел  
 static Scanner *scanner* = new Scanner(System.*in*);  
  
 // Инициализация пустого поля  
 public static void initializeField() {  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j < *SIZE*; j++) {  
 *field*[i][j] = -1; // Изначально все клетки пустые (обозначаем -1)  
 }  
 }  
 }  
  
 // Печать игрового поля  
 public static void printField() {  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j < *SIZE*; j++) {  
 if (*field*[i][j] == -1) {  
 System.*out*.print(".\t"); // Пустая клетка  
 } else {  
 System.*out*.print(*field*[i][j] + "\t"); // Заполненная клетка  
 }  
 }  
 System.*out*.println();  
 }  
 }  
  
 // Проверка, можно ли добавить число (остались ли свободные ячейки для числа)  
 public static boolean canAddNumber(int number) {  
 int index = Arrays.*binarySearch*(*NUMBERS*, number);  
 return *remainingCounts*[index] > 0;  
 }  
  
 // Получение случайного доступного числа  
 public static int getRandomNumber() {  
 Random random = new Random();  
 int index;  
 do {  
 index = random.nextInt(*NUMBERS*.length);  
 } while (*remainingCounts*[index] == 0); // Ищем число, которое можно использовать  
 *remainingCounts*[index]--;  
 return *NUMBERS*[index];  
 }  
  
 // Ход игрока (человека)  
 public static void playerMove() {  
 System.*out*.println("Ваш ход. Введите координаты (строка и столбец):");  
 int row, col;  
 do {  
 System.*out*.print("Строка (0-4): ");  
 row = *scanner*.nextInt();  
 System.*out*.print("Столбец (0-4): ");  
 col = *scanner*.nextInt();  
 } while (row < 0 || row >= *SIZE* || col < 0 || col >= *SIZE* || *filled*[row][col]); // Проверка на корректность хода  
  
 int number;  
 do {  
 System.*out*.print("Введите число от 0 до 13: ");  
 number = *scanner*.nextInt();  
 } while (number < 0 || number > 13 || !*canAddNumber*(number)); // Проверка, можно ли использовать это число  
  
 *field*[row][col] = number;  
 *filled*[row][col] = true; // Отмечаем клетку как заполненную  
 }  
  
 // Ход компьютера  
 public static void computerMove() {  
 System.*out*.println("Ход компьютера:");  
 Random random = new Random();  
 int row, col;  
 do {  
 row = random.nextInt(*SIZE*);  
 col = random.nextInt(*SIZE*);  
 } while (*filled*[row][col]); // Ищем свободную клетку  
  
 int number = *getRandomNumber*(); // Получаем случайное доступное число  
 *field*[row][col] = number;  
 *filled*[row][col] = true; // Отмечаем клетку как заполненную  
 System.*out*.println("Компьютер поставил " + number + " в клетку (" + row + ", " + col + ")");  
 }  
  
 // Подсчет очков за комбинации  
 public static int calculatePoints() {  
 int points = 0;  
  
 // Проверка на четыре одинаковых числа в строке  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j <= *SIZE* - 4; j++) {  
 if (*field*[i][j] == *field*[i][j + 1] && *field*[i][j] == *field*[i][j + 2] && *field*[i][j] == *field*[i][j + 3]) {  
 points += 160; // Награда за 4 одинаковых числа в ряду  
 }  
 }  
 }  
  
 // Можно добавить дополнительные проверки (столбцы, диагонали и т.д.)  
 return points;  
 }  
  
 // Основной игровой цикл  
 public static void playGame() {  
 *initializeField*(); // Инициализация пустого поля  
 boolean isPlayerTurn = true; // Ход игрока  
  
 // Игра продолжается, пока есть пустые клетки  
 while (true) {  
 *printField*(); // Печать текущего состояния поля  
  
 if (isPlayerTurn) {  
 *playerMove*(); // Ход игрока  
 } else {  
 *computerMove*(); // Ход компьютера  
 }  
  
 // Проверяем, остались ли свободные клетки  
 boolean hasEmptyCells = false;  
 for (boolean[] row : *filled*) {  
 for (boolean cell : row) {  
 if (!cell) {  
 hasEmptyCells = true;  
 break;  
 }  
 }  
 }  
 if (!hasEmptyCells) break; // Если нет свободных клеток, игра заканчивается  
  
 // Смена хода  
 isPlayerTurn = !isPlayerTurn;  
 }  
  
 // Подсчет очков после игры  
 int playerPoints = *calculatePoints*();  
 System.*out*.println("Игра завершена.");  
 *printField*(); // Печать финального состояния поля  
 System.*out*.println("Ваши очки: " + playerPoints);  
 }  
  
 public static void main(String[] args) {  
 *playGame*(); // Запуск игры  
 }  
}

Таблица 8 – Задание 8.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| SIZE | final int | Размер игрового поля (5x5). |
| NUMBERS | final int[] | Массив чисел от 0 до 13, которые используются для заполнения поля. |
| field | int[][] | Двумерный массив для хранения чисел, которыми заполняется игровое поле. |
| filled | boolean[][] | Двумерный массив, указывающий, заполнена ли клетка на поле (true - заполнена, false - нет). |
| counts | final int[] | Массив, задающий количество каждого числа от 0 до 13, которое можно использовать в игре. |
| remainingCounts | int[] | Копия массива counts, используемая для отслеживания оставшихся доступных чисел. |
| scanner | Scanner | Объект для ввода данных от игрока (используется для ввода координат и числа). |
| random | Random | Объект для генерации случайных чисел для хода компьютера. |
| row | int | Переменная для хранения координаты строки, вводимой игроком. |
| col | int | Переменная для хранения координаты столбца, вводимой игроком. |
| number | int | Число, которое вводит игрок или выбирает компьютер для размещения на поле. |
| isPlayerTurn | boolean | Флаг, указывающий, чей сейчас ход (true - ход игрока, false - ход компьютера). |
| playerPoints | int | Количество очков игрока, подсчитанных в конце игры. |
| hasEmptyCells | boolean | Флаг для проверки, остались ли на поле пустые клетки (true - есть пустые клетки). |

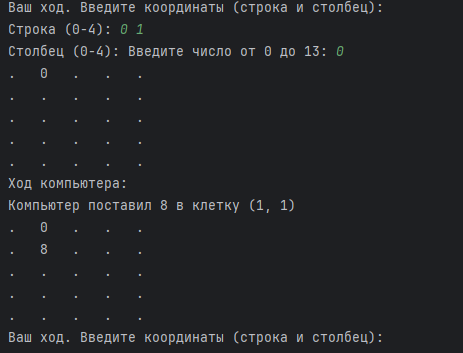


Рисунок 9 – Результат работы задания 9.

# 2. Разработка АИС

### 3.1 Описание и цели системы

Название: Автоматизированная Информационная Система (АИС) "Издательство"

Цель: Система предназначена для автоматизации учета информации об издательствах, авторах и книгах, выпускаемых данным издательством. АИС позволяет добавлять, редактировать и удалять данные, а также отображать информацию в удобной форме для сотрудников издательства.

### 3.2 Функциональные требования

* Учет издательств: Возможность добавления, редактирования и удаления данных об издательствах. Каждое издательство имеет уникальный идентификатор, название, адрес и контактную информацию.
* Учет авторов: Управление информацией об авторах, включая добавление, редактирование и удаление записей. Авторы идентифицируются по уникальному ID, имеют имя и краткую биографию.
* Учет книг: Учет информации о книгах, включая название, автора, издателя и цену. Каждая книга связана с конкретным издательством и автором.
* Связи между сущностями: Система должна обеспечивать корректное связывание книг с авторами и издательствами. Например, каждая книга связана с одним автором и издательством, но один автор может быть связан с несколькими книгами.

### 3.3 Нефункциональные требования

* Надежность: Система должна обеспечивать корректное хранение данных и целостность связей между сущностями.
* Безопасность: Введение уровней доступа для защиты данных от несанкционированного изменения.
* Удобство использования: Простота и интуитивность интерфейса.
* Масштабируемость: Возможность расширения системы при увеличении объема данных.
* Инструментальная платформа: Приложение реализовано на Java с использованием Swing для графического интерфейса и JUnit для тестирования.

### 3.4 Структура системы и описание классов

Классы:

1. Publisher: Содержит поля id, name, address, phone для хранения информации об издательстве.
2. Author: Включает id, name, biography для хранения данных об авторах.
3. Book: Хранит id, title, price, authorId, publisherId для описания книги и связи её с автором и издателем.
4. PublishingDatabase: Основной класс для управления данными. Включает методы для добавления, редактирования и удаления записей в таблицах "Издательства", "Авторы" и "Книги".

Основные методы:

* addPublisher(), editPublisher(), deletePublisher(): Управление данными издательств.
* addAuthor(), editAuthor(), deleteAuthor(): Управление данными авторов.
* addBook(), editBook(), deleteBook(): Управление данными книг и их связями с авторами и издательствами.

### 3.5 Алгоритмы и описание действий пользователя

Алгоритм добавления книги

1. Пользователь вводит название книги, выбирает автора и издательство, указывает цену.
2. Система проверяет, существуют ли выбранные автор и издательство.
3. Если данные корректны, книга добавляется в базу данных с уникальным идентификатором.

Алгоритм удаления книги

1. Пользователь выбирает книгу для удаления.
2. Система подтверждает, что книга связана только с одним автором и издательством.
3. После подтверждения система удаляет запись.

### 3.6 Тестирование системы

План тестирования

1. Функциональные тесты: Проверка добавления, редактирования и удаления данных для каждой сущности (издательства, авторы, книги).
2. Тесты на целостность данных: Проверка корректности связи между книгами, авторами и издательствами при редактировании и удалении данных.
3. Проверка безопасности: Тестирование ограничений для предотвращения некорректного ввода.

Основные сценарии тестирования

* Тест добавления нового издательства: Проверяет, что издательство с указанными параметрами успешно добавляется и доступно для просмотра.
* Тест добавления новой книги: Удостоверяется, что книга может быть добавлена только при указании корректных идентификаторов автора и издательства.
* Тест удаления книги: Убеждается, что при удалении книги все ссылки на неё корректно удаляются из базы данных.

### 3.7 Техническое задание (ТЗ)

Цель: Разработка системы для управления данными о книгах, авторах и издательствах с графическим интерфейсом и встроенной системой тестирования для обеспечения надежности.

Функциональные требования:

* Управление данными издательств, книг и авторов.
* Возможность выполнения CRUD-операций (Create, Read, Update, Delete).
* Поддержка целостности данных и правильности ссылок между сущностями.

Нефункциональные требования:

* Платформа: Java.
* Интерфейс: Swing GUI.
* Тестирование: JUnit.

Разграничение доступа:

* Два уровня пользователей: администратор (с правами редактирования данных) и пользователь (с правами просмотра).

### 3.8 Руководство пользователя

* Добавление данных: Для добавления данных в таблицу выберите нужный тип сущности и заполните поля в диалоге добавления.
* Редактирование данных: Выберите сущность в таблице и нажмите "Редактировать", чтобы открыть диалог редактирования.
* Удаление данных: Выберите запись в таблице и нажмите "Удалить" для её удаления.

Совет: Пользователь может просмотреть все записи в таблице, переключая фильтры и выбирая нужные данные для удобства работы.

### 3.9 Протокол тестирования

Таблица 9 – Задание 9.

|  |  |  |  |
| --- | --- | --- | --- |
| Тест-кейс | Описание | Ожидаемый результат | Статус |
| Добавление издательства | Добавление нового издательства | Издательство добавлено и отображается в таблице | Пройден |
| Редактирование автора | Редактирование данных автора | Данные автора обновлены | Пройден |
| Удаление книги | Удаление книги из базы данных | Книга удалена без нарушений ссылочной целостности | Пройден |

## Разработка Базы Данных

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.util.ArrayList;

import java.util.List;

// Класс для представления книги

class Book {

String title;

String author;

int year;

String publisher;

public Book(String title, String author, int year, String publisher) {

this.title = title;

this.author = author;

this.year = year;

this.publisher = publisher;

}

}

// Класс для представления автора

class Author {

String name;

String biography;

public Author(String name, String biography) {

this.name = name;

this.biography = biography;

}

}

// Класс для представления издательства

class Publisher {

String name;

String address;

public Publisher(String name, String address) {

this.name = name;

this.address = address;

}

}

public class PublishingDatabaseGUI extends JFrame {

private List<Book> books = new ArrayList<>();

private List<Author> authors = new ArrayList<>();

private List<Publisher> publishers = new ArrayList<>();

private DefaultTableModel bookTableModel;

private DefaultTableModel authorTableModel;

private DefaultTableModel publisherTableModel;

private JTable mainTable;

private JPanel buttonPanel;

public PublishingDatabaseGUI() {

super("Издательство: База данных");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(800, 600);

setLocationRelativeTo(null);

// Таблицы

bookTableModel = new DefaultTableModel(new Object[]{"Название", "Автор", "Год", "Издательство"}, 0);

authorTableModel = new DefaultTableModel(new Object[]{"Имя", "Биография"}, 0);

publisherTableModel = new DefaultTableModel(new Object[]{"Название", "Адрес"}, 0);

mainTable = new JTable(bookTableModel);

JScrollPane tableScrollPane = new JScrollPane(mainTable);

// Панель кнопок переключения таблиц

JPanel tableSwitchPanel = new JPanel();

JButton booksButton = new JButton("Книги");

JButton authorsButton = new JButton("Авторы");

JButton publishersButton = new JButton("Издательства");

tableSwitchPanel.add(booksButton);

tableSwitchPanel.add(authorsButton);

tableSwitchPanel.add(publishersButton);

// Панель кнопок действий

buttonPanel = new JPanel();

JButton addButton = new JButton("Добавить");

JButton editButton = new JButton("Редактировать");

JButton deleteButton = new JButton("Удалить");

buttonPanel.add(addButton);

buttonPanel.add(editButton);

buttonPanel.add(deleteButton);

// Добавляем элементы в главное окно

getContentPane().add(tableSwitchPanel, BorderLayout.NORTH);

getContentPane().add(tableScrollPane, BorderLayout.CENTER);

getContentPane().add(buttonPanel, BorderLayout.SOUTH);

// Переключение между таблицами

booksButton.addActionListener(e -> switchToBooks());

authorsButton.addActionListener(e -> switchToAuthors());

publishersButton.addActionListener(e -> switchToPublishers());

// Обработчики кнопок действий

addButton.addActionListener(e -> handleAddAction());

editButton.addActionListener(e -> handleEditAction());

deleteButton.addActionListener(e -> handleDeleteAction());

setVisible(true);

}

// Переключение на таблицу книг

private void switchToBooks() {

mainTable.setModel(bookTableModel);

}

// Переключение на таблицу авторов

private void switchToAuthors() {

mainTable.setModel(authorTableModel);

}

// Переключение на таблицу издательств

private void switchToPublishers() {

mainTable.setModel(publisherTableModel);

}

// Обработка кнопки "Добавить"

private void handleAddAction() {

DefaultTableModel currentModel = (DefaultTableModel) mainTable.getModel();

if (currentModel == bookTableModel) {

addBook();

} else if (currentModel == authorTableModel) {

addAuthor();

} else if (currentModel == publisherTableModel) {

addPublisher();

}

}

private void addBook() {

JTextField titleField = new JTextField();

JTextField authorField = new JTextField();

JTextField yearField = new JTextField();

JTextField publisherField = new JTextField();

Object[] fields = {

"Название:", titleField,

"Автор:", authorField,

"Год:", yearField,

"Издательство:", publisherField

};

int option = JOptionPane.showConfirmDialog(this, fields, "Добавить книгу", JOptionPane.OK\_CANCEL\_OPTION);

if (option == JOptionPane.OK\_OPTION) {

String title = titleField.getText();

String author = authorField.getText();

int year = Integer.parseInt(yearField.getText());

String publisher = publisherField.getText();

books.add(new Book(title, author, year, publisher));

bookTableModel.addRow(new Object[]{title, author, year, publisher});

}

}

private void addAuthor() {

JTextField nameField = new JTextField();

JTextField biographyField = new JTextField();

Object[] fields = {

"Имя автора:", nameField,

"Биография:", biographyField

};

int option = JOptionPane.showConfirmDialog(this, fields, "Добавить автора", JOptionPane.OK\_CANCEL\_OPTION);

if (option == JOptionPane.OK\_OPTION) {

String name = nameField.getText();

String biography = biographyField.getText();

authors.add(new Author(name, biography));

authorTableModel.addRow(new Object[]{name, biography});

}

}

private void addPublisher() {

JTextField nameField = new JTextField();

JTextField addressField = new JTextField();

Object[] fields = {

"Название издательства:", nameField,

"Адрес издательства:", addressField

};

int option = JOptionPane.showConfirmDialog(this, fields, "Добавить издательство", JOptionPane.OK\_CANCEL\_OPTION);

if (option == JOptionPane.OK\_OPTION) {

String name = nameField.getText();

String address = addressField.getText();

publishers.add(new Publisher(name, address));

publisherTableModel.addRow(new Object[]{name, address});

}

}

private void handleEditAction() {

JOptionPane.showMessageDialog(this, "Функция редактирования пока не реализована.");

}

private void handleDeleteAction() {

int selectedRow = mainTable.getSelectedRow();

if (selectedRow == -1) {

JOptionPane.showMessageDialog(this, "Выберите запись для удаления.");

return;

}

DefaultTableModel currentModel = (DefaultTableModel) mainTable.getModel();

if (currentModel == bookTableModel) {

books.remove(selectedRow);

} else if (currentModel == authorTableModel) {

authors.remove(selectedRow);

} else if (currentModel == publisherTableModel) {

publishers.remove(selectedRow);

}

currentModel.removeRow(selectedRow);

}

public static void main(String[] args) {

SwingUtilities.invokeLater(PublishingDatabaseGUI::new);

}

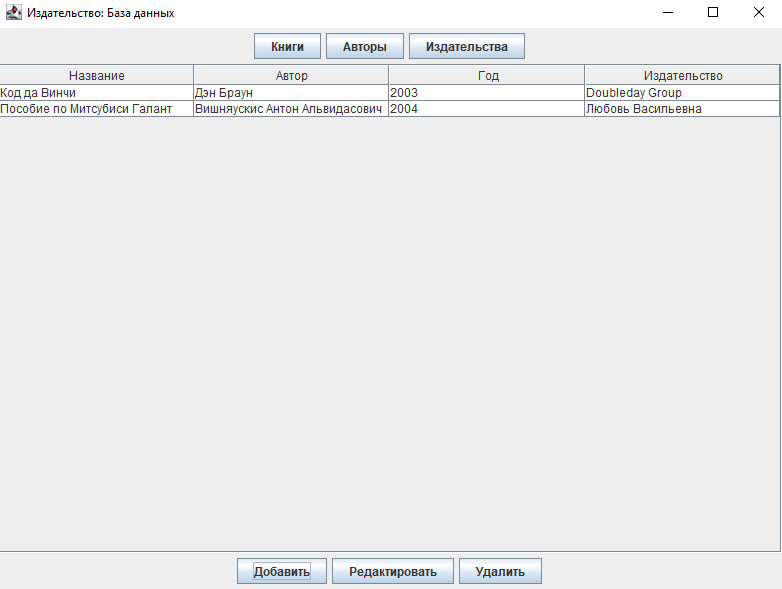
}

Рисунок 10 – Результат работы базы данных на Java.

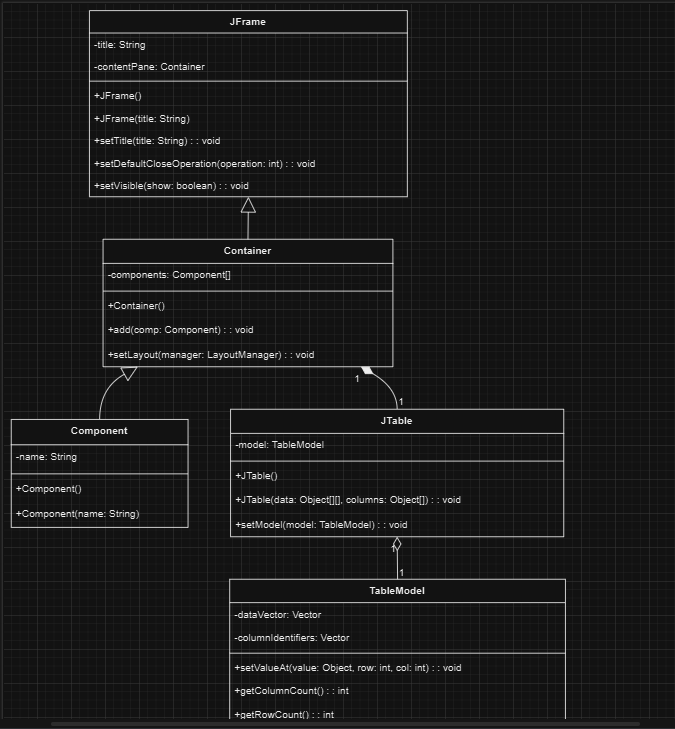


Рисунок 11 – Диаграмма работы базы данных.

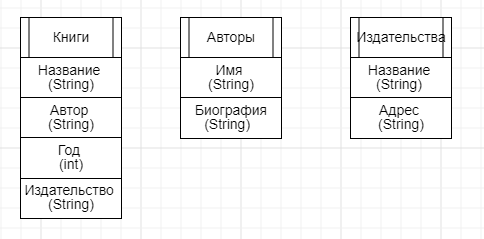


Рисунок 12 – Диаграмма таблиц в базе данных.

## Разработка 1С «Издательство»

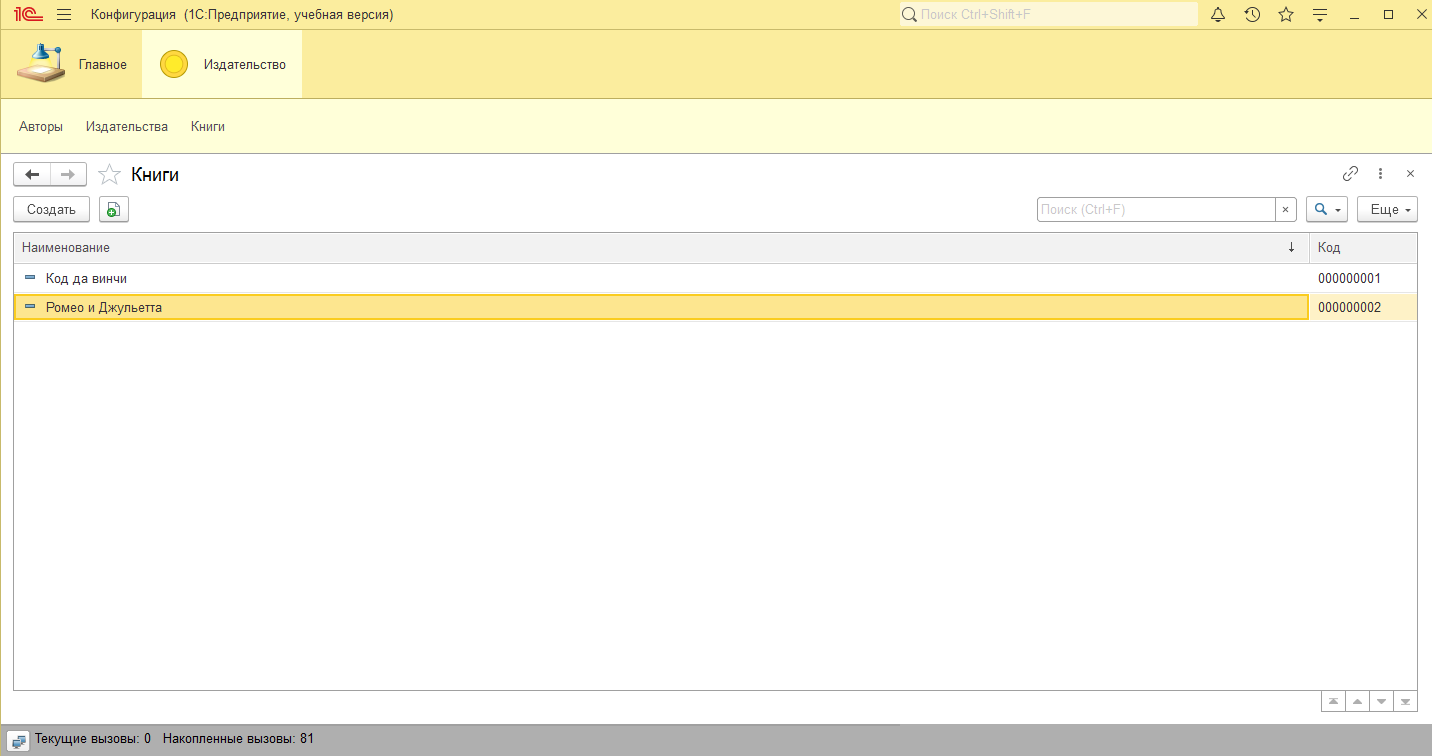
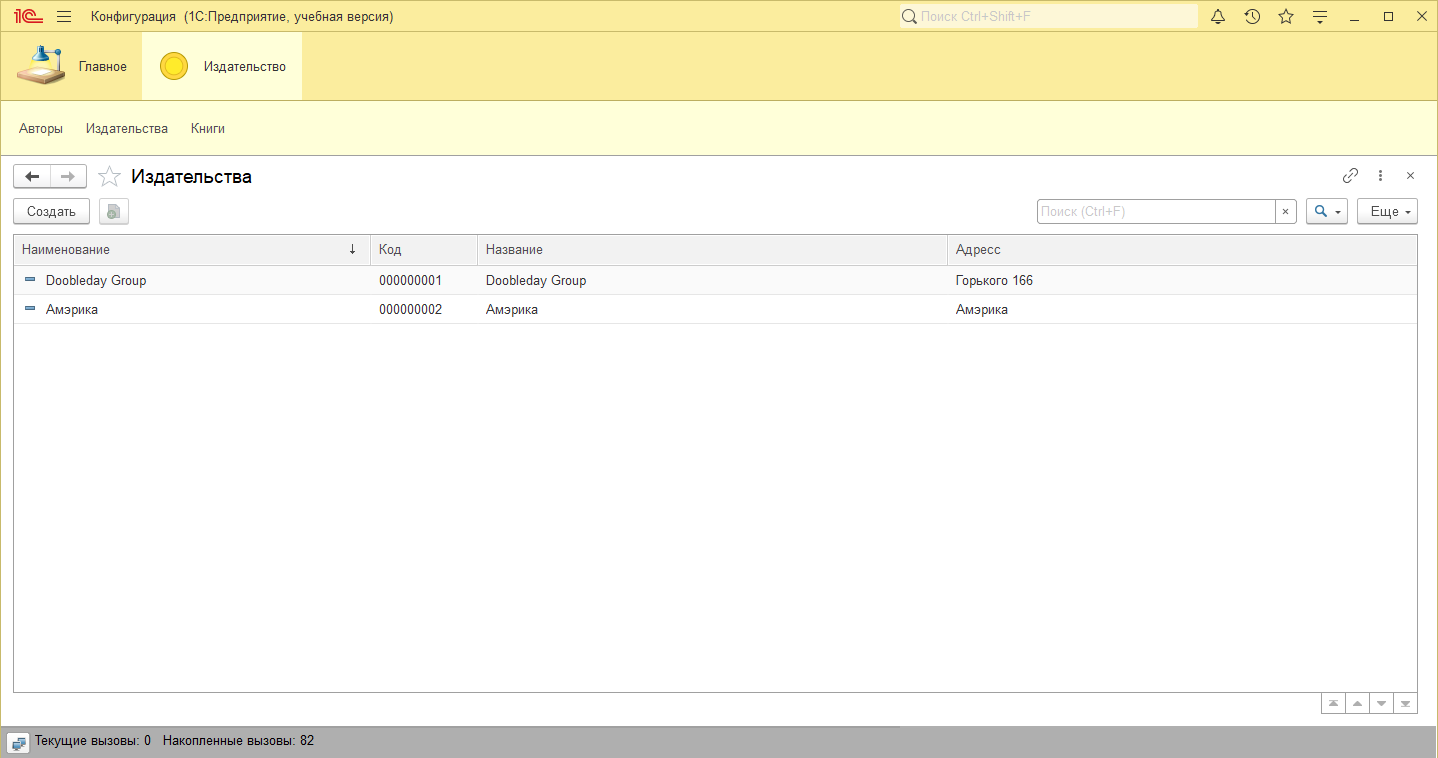


Рисунок 13 – База данных «Издательство», таблица «Книги».

Рисунок 14 – Таблица «Издательства»

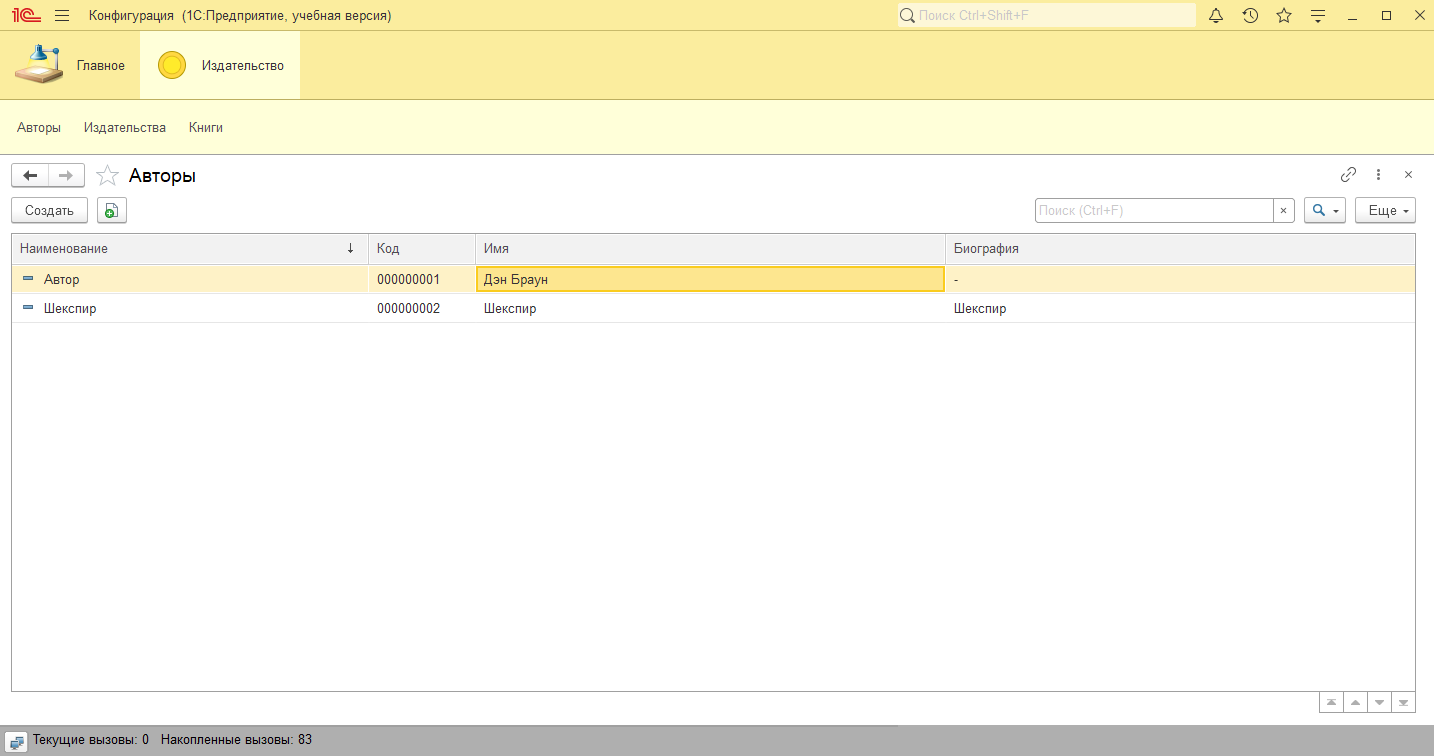


Рисунок 15 – Таблица «Автор»

Структура базы данных:

Справочник "Книги":

Поля: Название, Автор, Год, Издательство.

Справочник "Авторы":

Поля: Имя, Биография.

Справочник "Издательства":

Поля: Название, Адрес.

Связи:

Книги → Авторы (ссылка).

Книги → Издательства (ссылка).

## Разработка мобильного приложения

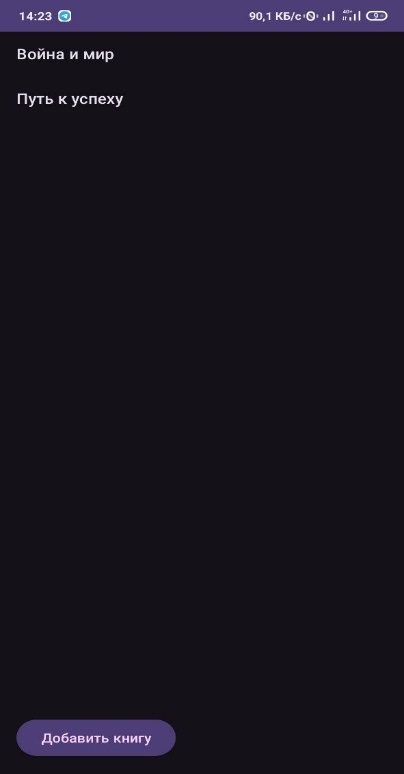


Рисунок 15 – Главный интерфейс приложения.

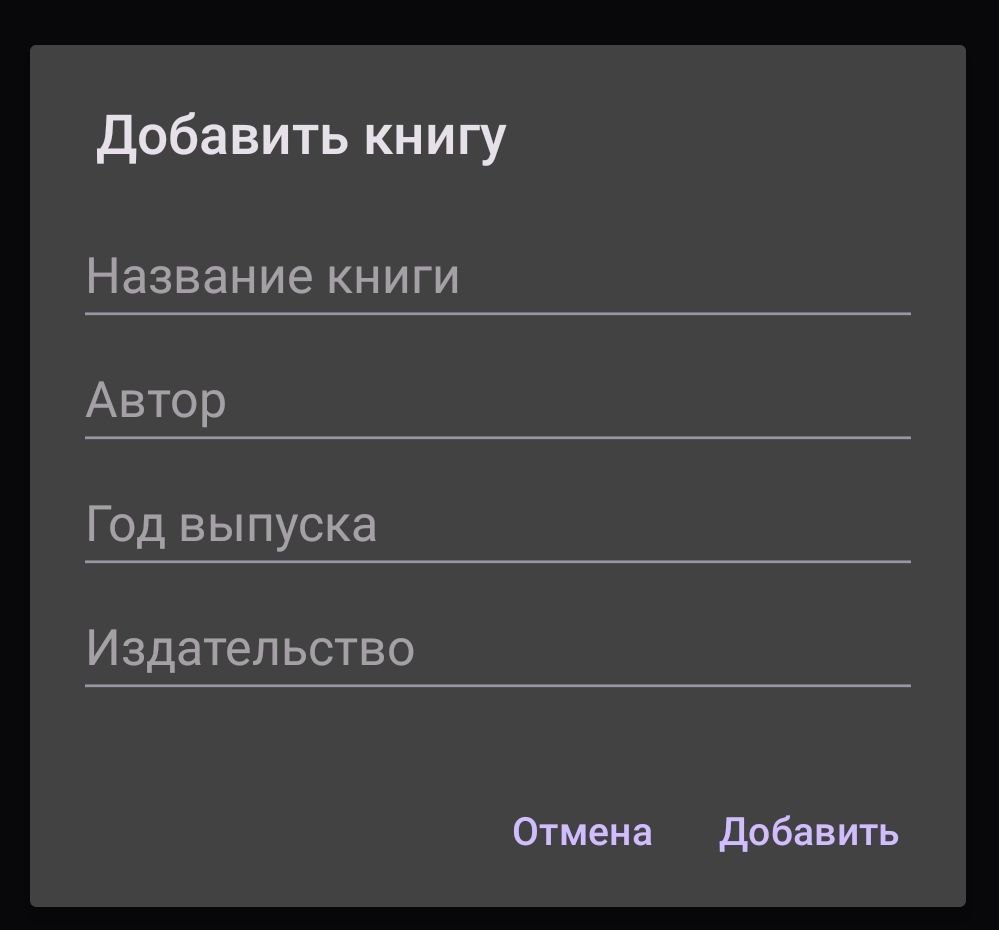


Рисунок 17 - Кнопка добавить открывает окно в котором можно добавить книгу и данные про нее.

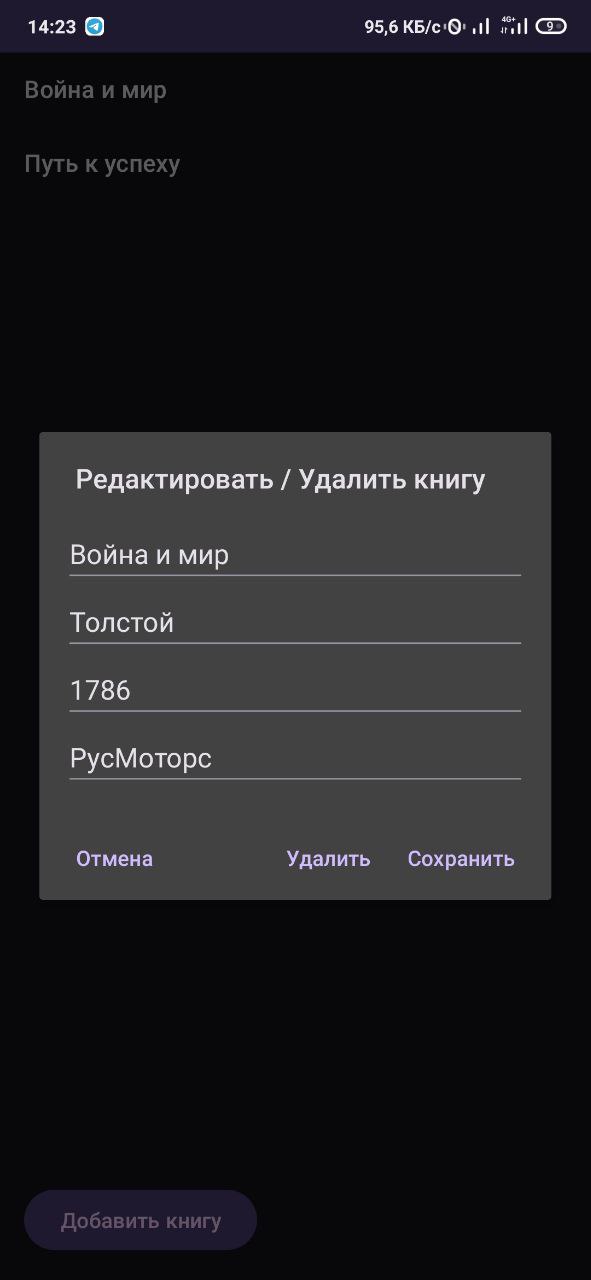


Рисунок 16 – Редактирование книги(вызывается долгим нажатием на книгу).

Сайт: <https://vi2635.craftum.io/>

Ссылка на гитхаб: https://github.com/hallabes/practice

Заключение  
В рамках практики были разработаны программные модули, создано мобильное приложение, проведено тестирование и поддержка кода, а также решены задачи в области системного программирования. Задания выполнены в соответствии с требованиями, а результаты работы представлены на сайте, созданном для демонстрации проделанной работы. Практика позволила не только углубить теоретические знания, но и приобрести важный практический опыт, необходимый для дальнейшего профессионального развития в сфере программирования.

# Список используемой литературы

1. Смирнов, Д. С. Базы данных и системы управления ими: учебное пособие для вузов / Д. С. Смирнов, В. И. Поляков. – Москва: Академия, 2021. – 240 с.
2. Михайлов, А. В. Программирование на Java для начинающих: учебное пособие / А. В. Михайлов, Л. Г. Козлова. – Санкт-Петербург: Лань, 2023. – 320 с.
3. Иванова, Е. П. Разработка мобильных приложений для Android: учебное пособие / Е. П. Иванова. – 3-е изд., перераб. – Санкт-Петербург: Питер, 2022. – 280 с.
4. Соколова, О. Н. Основы мобильной разработки на платформе Android: учебное пособие для студентов / О. Н. Соколова, Р. П. Исаев. – Москва: КноРус, 2020. – 210 с.
5. Лебедев, С. Н. Основы работы с базами данных в MySQL: учебное пособие / С. Н. Лебедев. – Новосибирск: НГТУ, 2022. – 195 с.
6. Сидоров, Ю. Б. Введение в мобильную разработку на Android: учебное пособие / Ю. Б. Сидоров, А. В. Кузнецов. – Москва: РИПОЛ классик, 2021. – 158 с.
7. Трофимов, П. И. Проектирование баз данных: учебное пособие / П. И. Трофимов, И. А. Захаров. – Екатеринбург: УрФУ, 2019. – 185 с.
8. Беляева, Н. М. Программирование и разработка приложений на Java: учебное пособие / Н. М. Беляева. – Москва: Форум, 2023. – 276 с.
9. Петрова, Е. Л. Управление данными в современных базах данных: учебное пособие / Е. Л. Петрова, И. О. Коваленко. – Казань: КазГТУ, 2020. – 208 с.
10. Филиппов, К. Р. Android-разработка: руководство для начинающих / К. Р. Филиппов, Л. А. Головин. – Санкт-Петербург: БХВ-Петербург, 2023. – 310 с.